Smoldyn
A Spatial Stochastic Simulator

Version 1.72, © February 2007

Steven Andrews

1. Overview of *Smoldyn*

Smoldyn is a computer program which simulates the dynamics of reaction-diffusion systems on a microscopic scale. While the code has been written to be platform independent, most recent development has been done on a Macintosh, running OS 10.4.8. It has also been compiled on Linux and Windows systems. OpenGL support (which is standard with most modern operating systems) is required for graphical output.

Molecules in the simulation are represented individually, diffuse within a rectanguloid volume by Brownian motion, and undergo simple chemical reactions. The timescale considered is short enough that diffusion within the simulation volume is explicitly modeled using exact molecular positions, but is long enough that momenta and molecular orientations can be assumed to take on average values. This is often called the Smoluchowski level of approximation (and hence the name of the program). A further approximation is that molecules do not occupy volume. Time steps are synchronous, alternating steps in which molecules diffuse and those in which they react. Space is defined as an arbitrary dimensional rectanguloid volume which has fixed walls. The walls can be reflective, absorbing, or periodic. Chemical reactions may be zeroth order, unimolecular, or bimolecular, and may have any number of products. It is now possible to include surfaces that can be composed from several standard shapes, where these surfaces can be reflecting, absorbing, or transparent. The algorithms for *Smoldyn* are described in a research paper written by Dennis Bray and myself titled "Stochastic simulation of chemical reactions with spatial resolution and single molecule detail" (*Phys. Biol.*, 1:137-151, 2004).

2. The Configuration File

2.1 Starting *Smoldyn*

Before running *Smoldyn*, the parameters for a simulation need to be written in a text format configuration file and saved to disk. Some sample configuration files should have been included with the executable program, one of which is presented in section 2.5 of this documentation. When *Smoldyn* is run, the program asks for the name of the configuration file and for a few additional parameters; then it loads the file, runs the simulation, and terminates. The only user input that is processed during the simulation is for simple graphics manipulation and simulation pausing.

The name of the configuration file is typically given to *Smoldyn* after the program is running, when it asks for the name of a configuration file. At the text prompt, type in the configuration file name, including file path information. Spaces are allowed in the file name or file path. Here are some examples, using a configuration file called `config` and Macintosh file path notation:

`config` is in same directory as *Smoldyn*:	`config`
`config` is in a subdirectory of where *Smoldyn* is:	`:folder:config`
Absolute path name, starting from the volume:	`Mac HD:files:folder:config`

It should also be possible to have a relative file path that ascends the file tree, as well as descending the tree, although I have not figured out how to do this with the Macintosh file system yet. Standard file path notation should also work with PCs or with Linux, but have not been tested.

After the configuration file name is entered, *Smoldyn* asks for runtime flags, where options are the characters '`q`', '`p`', '`-`', or a combination of characters. '`q`' is used for quiet operation, in which no diagnostics or parameters are displayed, and is minimally useful. '`p`' indicates that the simulation should be set up and that all diagnostics and parameters should be calculated and displayed, but then the actual simulation should not be run. This is useful for choosing appropriate parameters for complex simulations. A '`-`' is used to indicate that neither the '`q`' nor '`p`' options are desired.

It should also be possible to run *Smoldyn* from a Unix style command line, in which case the name of the configuration file and the runtime flags can be entered on the same command line, thus removing the need for text entry during program execution. In this case, the order of the file name and the flags is unimportant and any runtime flags need to be preceded by a '`-`'. I have not figured out how to do this yet using the Macintosh system.

With the exception of graphics, which cannot be saved, all output from *Smoldyn* is saved to text files, allowing their analysis with a wide variety of other software. These output file names are declared in the configuration file, as explained below. These output files can be saved in the same file folder as the configuration file, or in a sub-directory of that folder.

2.2 Configuration file format

The design of a simulation can be broken down into two portions. One portion includes the parameters of the physical system, including its shape, the molecules that are present, the reactions that take place, and any surfaces. These parameters are entered in the configuration file using a variety of statements and are simulated using the core program. Here are some examples of statements:

`dim 3`	3 dimensional system
`high_wall 0 100 r`	properties of a system boundary
`time_start 0`	starting time for the simulation
`mol 10 CheY 50 50 50`	creation of 10 CheY molecules

The other portion of a simulation is the action of the experimenter, which includes measurements of the system and external perturbations to the system. These actions are listed, also in the configuration file, with a series of commands with execution times; they are not considered by *Smoldyn* until they are supposed to happen. When a command is supposed to be executed, *Smoldyn* processes it with a runtime command interpreter, which is an auxiliary portion of the program that is designed to be easily modified. Here are some examples of commands:

`cmd e ifno asp stop`	conditional command, run at every iteration

`cmd @ 20 molcount outfile` molecules are counted and recorded at time 20

Simulation parameters need to be entered using the formats shown in the table given below. Formatting errors should be caught by *Smoldyn*, causing a program termination and, hopefully, a useful error message. On each line of input, *Smoldyn* reads a word that tells what the line contains and then reads the number of items that it expects. Both too few items and too many items cause errors, although it is always possible to add comments to the end of a line using a number sign. In many cases, lines may entered in any order, although some basic definitions need to be entered near the top of the file. Default values are used for parameters that are not defined in the configuration file. While many instructions can only be entered once, such as the system dimensionality, others can be entered multiple times, such as lines to define various collections of molecules.

Reactions are entered as a block of definitions, beginning with the word "`start_reaction`" and ending with "`end_reaction`", between which only instructions that are relevent to reactions are allowed. Reactions are stored internally with two fundamental parts: a reactant table and a list of reactions, including both rates and products. The reactant table associates reactants with numbered reactions but does not give further details about them; in effect it is only the set of species on the left side of the arrows in a list of reactions. The reaction list includes rate constants and lists of products; in effect it is the set of species on the right side of the arrows. This structure is used in the reaction section of the input file as well. A confusing aspect of reactions is that reversible reactions (and some so-called continuation reactions) require a parameter from which *Smoldyn* can figure out where to place multiple reaction products to prevent immediate recombination. This issue is explained in section 3 of this documentation.

Surfaces are also entered with blocks of definitions. A surface may be made of up many panels. In many cases, these panel will be adjacent, such as for a triangulated mesh surface, the sides of a box, or a bacterium that is composed of a cylinder plus hemispherical endcaps. However, the panels can also be non-contiguous. Each surface has a uniform drawing color and method, and also interacts with molecules the same way on each panel. These interactions can be transparent, reflective, or absorbing, which can be set the same for all molecules, or can be different for different molecules. Currently, there is no support for either surface-bound molecules, nor for "leaky" surfaces, but both of these are being considered for future additions.

Since *Smoldyn* does not use any particular set of units, it is up to the user to make sure units are consistent. Some useful conversion factors are:

$10^{-6}\ \mathrm{cm^2s^{-1}} = 10^{-10}\ \mathrm{m^2s^{-1}} = 100\ \mathrm{\mu m^2s^{-2}} = 0.1\ \mathrm{\mu m^2ms^{-1}} = 100\ \mathrm{nm^2\mu s^{-1}} = 0.1\ \mathrm{nm^2ns^{-1}}$

$1\ \mathrm{M^{-1}s^{-1}} = 10^{-3}\ \mathrm{m^3mol^{-1}s^{-1}} = 1.66\mathrm{e}{-27}\ \mathrm{m^3s^{-1}} = 1.66\mathrm{e}{-18}\ \mathrm{\mu m^3ms^{-1}} = 1.66\mathrm{e}{-9}\ \mathrm{nm^3ns^{-1}}$

2.3 Configuration file statements

<u>Statements about the configuration file</u>

# *text*

Single-line comment. A '`#`' symbol indicates that the rest of the line is a comment.

`/*`
text
`*/`

Multi-line comment. All lines between "/*" and the following "*/" are ignored. These must be the first "words" on a line. Additional text on these lines is ignored as well.

`read_file` *filename*

Read some other configuration file, returning to the present one when that one has been read.

`end_file`

End of configuration file. This line is optional, as *Smoldyn* can also just read until the file ends.

<u>Definition of system parameters</u>

`dim` *dim*

Dimensionality of the system. Must be at least one, and is typically between 1 and 3. Larger numbers are permitted as well.

`max_names` *int*

Maximum number of molecular species that will be used. It is permissable to use this command and one or more "`name`" commands, or "`names`" but not both.

`name` *name*

Name of a molecule. Standard naming conventions are followed, in that the name should start with a letter and spaces are not permitted. This command must be used in combination with "`max_names`".

`names` $name_1$ $name_2$ … $name_n$

Names of all of the types of molecules present in the system. Standard naming conventions are followed, in that the name should start with a letter and spaces are not permitted. The command "`max_names`" cannot be used if this is used.

`difc` *name float*

Isotropic diffusion constant of molecule type *name*. Default value is 0.

`difm` *name* $float_0$ $float_1$ … $float_{dim*dim-1}$

Square root of diffusion matrix of *name* (the dot product of this matrix and itself is the anisotropic diffusion matrix). The matrix has dim^2 terms (*dim* is the system dimensionality), listed row by row of the matrix; the matrix is supposed to be symmetric. If this line is not entered, isotropic diffusion is assumed, which leads to a faster runtime. While a matrix is used for diffusion if one is given, the value stored with `difc` is used for reaction rate calculations. If `difc` is not entered, the trace of the square of this matrix, divided by the system dimensionality, is used as a proxy for the isotropic diffusion coefficient to allow reaction rates to be estimated. This line is most useful for restricting diffusion to a plane or a line, in which case the square root of the diffusion coefficient is given for each diagonal element of the matrix where there is diffusion and 0s are place on diagonal elements for axes where diffusion is not possible, as well as on off-diagonal elements.

`low_wall` *dim pos type*

Creates a lower boundary for the simulation volume. This wall is perpendicular to the dimension *dim* such that all locations between *pos* and the position of the corresponding upper boundary are considered to be within the simulation volume. The type of wall is given in *type*, which should be one of four single letter codes: '`r`' means a reflecting wall, '`p`' means a periodic wall (also called wrap-around or toroidal), '`a`' means an absorbing wall, and '`t`' means a transparent wall. Transparent walls imply an unbounded system and may lead to slow simulations. *If any surfaces are defined for the simulation, then walls still must be entered to define the system volume, but these walls are essentially non-functional (the sole exception is that reactions can occur across periodic walls). Additional surfaces need to be defined to serve as the system boundaries.*

`high_wall` *dim pos type*

Identical to the definition for `low_wall`, although this creates the upper boundary for the simulation volume. See note about surfaces in `low_wall`.

`max_mol` *int*

Maximum possible number of molecules for which memory should be allocated. The simulation terminates if more molecules are required than are allocated initially. Note that most reactions require a few extra molecule spaces for processing.

`mol` *nmol name* pos_0 pos_1 … pos_{dim-1}

Simulation starts with *nmol* type *name* molecules at location *pos*. Each of the *dim* elements of the position may be a number to give the actual position of the molecule or molecules; or the letter '`u`' to indicate that the position for each molecule should be a random value between the bounding walls, chosen from a uniform density; or a position range which is given as two numbers separated with a hyphen.

<u>Simulation performance statements</u>

`rand_seed` *int*

Seed for random number generator. If this line is not entered, the current time is used as a seed, producing different sequences for each run.

`accuracy` *float*

A parameter that determines the quantitative accuracy of the simulation, on a scale from 0 to 10. Low values are less accurate but run faster. Default value is 10, for maximum accuracy. When accuracy is 0, bimolecular reactions are only checked for pairs of reactants that are both within the same virtual box; with higher accuracy values, reactants in nearest neighboring boxes are considered as well, and then when accuracy is 10, reactants in all types of neighboring boxes are checked.

`molperbox` *float*

Virtual boxes are set up initially so the average number of molecules per box is no more than this value. The default value is 5. `boxsize` is an alternate way of entering comparable information.

`boxsize` *float*

Rather than using `molperbox` to specify the sizes of the virtual boxes, `boxsize` can be used to request the width of the boxes. The actual box volumes will be no larger than the volume calculated from the width given here.

Graphical display statements

`graphics` *str*

Type of graphics to use during the simulation. The options are '`none`' for no graphics, '`opengl`' for basic and fast OpenGL graphics, '`opengl_good`' for high quality and slow OpenGL graphics. If this line is not entered, no graphics are shown.

`graphic_iter` *int*

Number of time steps that should be run between each update of the graphics. Default value is 1.

`tiff_iter` *int*

Number of time steps that should be run between each automatic saving of a TIFF file. Default value is 0, meaning that TIFFs should not be saved automatically.

`tiff_name` *name*

Root filename for TIFF files, which may include path information if desired. Default is "`OpenGL`", which leads to the first TIFF being saved as "`OpenGL001.tif`".

`tiff_min` *int*

Initial suffix number of TIFF files that are saved. Default value is 1.

`tiff_max` *int*

Largest possible suffix number of TIFF files that are saved. Once this value has been reached, additional TIFFs cannot be saved. Default value is 999.

`frame_thickness` *int*

Thickness of the frame that is drawn around the simulation volume, in points. Default value is 2.

`grid_thickness` *int*

Thickness of the grid lines that can be drawn to show the virtual boxes. Default value is 0, so that the grid is not drawn.

`background_color` *red green blue*
`background_color` *red green blue alpha*

Color of the background. All values should be between 0 and 1 (use all 0s for black and all 1s for white (default). The *alpha* value is optional and may not work anyhow.

`display_size` *name float*

Size of molecule of type *name* for display to the graphical output. The default value is 3, indicating that each molecule is displayed with a small square; 0 indicates that a molecule should not be displayed and larger numbers yield larger squares.

`color` *name red green blue*

Red, green, and blue values for displaying molecules of type *name*. Each value should be between 0 and 1. Default values are 0 for each parameter, which is black. Some useful colors: black is 0 0 0, brown is 0.6 0.6 0, red is 1 0 0, orange is 1 0.7 0, yellow is 0.8 0.9 0, green is 0 1 0, blue is 0 0 1, violet is 0.6 0 0.6, grey is 0.4 0.4 0.4, white is 1 1 1, and blue-green is 0 0.6 0.5.

<u>Simulation time statements</u>

`time_start` *float*

Starting point for simulated time.

`time_stop` *float*

Stopping time of simulation, using simulated time. The simulation continues past the `time_stop` value by less than one time step.

`time_step` *float*

Time step for the simulation. Longer values lead to a faster runtime, while shorter values lead to higher accuracy. Also, longer values lead to bimolecular reactions that behave more as though they are activation limited, rather then diffusion limited.

`time_now` *float*

Another starting time of simulation. Default value is equal to `time_start`. If this time is before `time_start`, the simulation starts at `time_start`; otherwise, it starts at `time_now`.

Statements about the runtime command interpreter

`output_root` *str*

Root of path where text output should be saved. Spaces are permitted. Output files are saved in the same folder as the configuration file, modified by this string. See the description for `output_files`. Make sure that the destination folder has been created and that the string is terminated with a colon (and started with a colon if needed).

`output_files` str_1 str_2 … str_n

Declaration of filenames that can be used for output of simulation results. Spaces are not permitted in these names. Any previous files with these names will be overwritten. The path for these filenames starts from the configuration file and may be modified by a root given with `output_root`. For example, if the configuration file was called with `:folder:config.txt` and `output_root` was not used, then the output file `out.txt` will appear in the folder `folder` too. If the configuration file was called with `:folder:config.txt` and the output root was given as `results:`, then the output file goes to the `results` sub-folder of the folder `folder`. The filename “`stdout`” results in output being sent to the standard output. In most cases, it is also permissible to not declare filenames, in which case output is again sent to the standard output.

`output_file_number` *int*

Starting number of output file name. The default is 0, meaning that no number is appended to a name (*e.g.* the file name `out.txt` is saved as `out.txt`). A value larger than 0 leads to an appended file name (if 1 is used, then `out.txt` is actually saved as `out_001.txt`). Note that the command `incrementfile` increments the file number before it runs the rest of the command.

`cmd b,a,e` *string*
`cmd @` *time string*
`cmd n` *int string*
`cmd i` *on off dt string*

Declaration of a command to be run by the run-time interpreter, where the final portion labeled *string* is the actual command. The character following `cmd` is the command type, which may be ‘`b`’ for before the simulation, ‘`a`’ for after the simulation, ‘`e`’ for every time step during the simulation, ‘`@`’ for a single command execution at time *time*, ‘`n`’ for every *n*’th iteration of the simulation, or ‘`i`’ for a fixed time interval. For type ‘`i`’, the command is executed over the period from *on* to *off* with intervals of at least *dt* (the actual intervals will only end at the times of simulation time steps). See section 2.4 for the commands that are available.

`max_cmd` *int* (obsolete statement)
Maximum length of command queue. Default value is 10. As of version 1.55, this statement is no longer needed in configuration files, because the command queue is now expanded as needed.

<u>Reaction definitions</u>

`start_reaction`
Start of reaction definition. Between this instruction and “`end_reaction`”, all lines need to pertain to this order of reaction. It is permissible to list reactions of the same order in multiple blocks, provided that only the first block includes a `max_rxn` statement and that sufficient reactions are declared with that statement.

`order` *int*
Order of the reactions being declared (0, 1, or 2).

`max_rxn` *max_rxn*
Maximum number of reactions that will be declared of the given order.

`reactant` r_0 r_1 … r_{nrxn-1}
`reactant` *name* r_0 r_1 … r_{nrxn-1}
`reactant` $name_1$ `+` $name_2$ r_0 r_1 … r_{nrxn-1}
Declaration of reactants and reaction names for zeroth order, unimolecular, and bimolecular reactions, respectively. The listed molecule names are the reactants and the following strings are the respective reaction names. Note that there are spaces before and after the ‘`+`’ symbol.

`rate` *r rate*
Reaction rate constant for reaction called *r*. Units for the reaction rate constant are (volume)$^{order-1}$ times inverse time. These rates are converted by the program into probabilities or binding radii. To enter the simulation parameters directly, use `rate_internal`.

`rate_internal` *r float*
Internal value for reaction rate information, which can be used to override the internal rates that are calculated from the `rate` entry. For zeroth order reactions, this is the expectation total number of reactions per time step;

for unimolecular reactions, this is the reaction probability per time step for each reactant molecule; and for bimolecular reactions, this is the binding radius.

`product` *r name* + *name* + … + *name*

List of products for reaction *r*. Note that there are spaces before and after each '`+`' symbol.

`product_param` *r* `i`
`product_param` *r* `p,x,r,b,q,y,s` *float*
`product_param` *r* `o,f` *prod_name* pos_0 pos_1… pos_{dim-1}

Parameters for the initial placement of products of reaction *r*. A product parameter also affects the binding radius of the reverse reaction. These are explained in section 3. In the first format, a type of '`i`' indicates that the reverse reaction is ignored for calculations. The second format uses one of the type letters shown: '`p`' and '`q`' are geminate rebinding probabilities, '`x`' and '`y`' are maximum geminate rebinding probabilities, '`r`' and '`s`' are ratios of unbinding to binding radii, and '`b`' is a fixed unbinding radius. The third format yields products that have a fixed relative orientation, which is either randomly rotated with '`o`', or not rotated with '`f`'. In the absence of better information, a useful default parameter type is either '`x`' or '`y`', with a value of about 0.2.

`end_reaction`

End of reaction definition. Reaction instructions are no longer recognized but other simulation instructions are.

Surface definitions

`max_surface` *int*

Maximum number of surfaces that will be defined. Each surface may have many panels, including disjoint panels.

`start_surface`

Start of surface definition block. Between this instruction and "`end_surface`", all lines need to pertain to this surface. It is permissible to list parameters of one surface in multiple blocks.

`name` *name*

Name of this surface. If the name has not been used yet for a surface, then a new surface is started. All lines within this surface block pertain to this surface.

`action_front` *molec action*

The behavior of molecules named *molec* when they collide with the front of this surface. If *molec* is "all", then this action applies to all molecules. The action is a single letter, which is 'r' for reflection, 'a' for absorption, 't' for transparent, or 'p' for periodic. The default is reflection for all molecules. Periodic surfaces only work for rectangle panels and are only intended to be used for periodic boundaries of the system volume.

`action_back` *molec action*

The behavior of molecules named *molec* when they collide with the back of this surface. See `action_front` for details.

`action_both` *molec action*

The behavior of molecules named *molec* when they collide with either the front or back of this surface. See `action_front` for details.

`color_front` *red green blue*
`color_front` *red green blue alpha*

Color of the front of the surface. All color values are numbers between 0 and 1, where 1 is maximum intensity a 0 is minimum (1 1 1 is white). The *alpha* value is optional and describes the opacity of the surface. If entered, it also needs to be between 0 and 1, where 1 is an opaque object (the default) and 0 is transparent. OpenGL graphics do not work well with non-zero alpha values, so don't expect good results.

`color_back` *red green blue*
`color_back` *red green blue alpha*

Color of the back of the surface. See `color_front` for details.

`color` *red green blue*
`color` *red green blue alpha*
`color_both` *red green blue*
`color_both` *red green blue alpha*

Color of the front and back of the surface. See `color_front` for details.

`thickness` *float*

Boldness of the surface in pixels for drawing purposes. This is only relevent for 1-D and 2-D simulations, and for 3-D simulations in which surfaces are drawn with just vertices or edges and not faces. The simulated thickness of surfaces is given with the command thick.

`polygon_front` *char*

Drawing method for the front of the surface. The character may be 'f', for filling in the faces of the polygons (this is the default), 'e' for drawing the edges, 'g' for drawing both faces and edges, or 'v' for drawing the vertices.

`polygon_back` *char*

Drawing method for the back of the surface. See `polygon_front` for details.

`polygon_both` *char*

Drawing method for the front and back of the surface. See `polygon_front` for details.

`max_panels` *char int*

Maximum number of panels of shape *char* that will be defined for this surface. The shape may be 'r' for a rectangle, 't' for a triangle, or 's' for a sphere. The surface can include panels with different shapes.

`panel` *char float … float*

Defines a new panel for the surface, where the panel has shape *char*. The shape may be 'r' for a rectangle, 't' for a triangle, 's' for a sphere, 'c' for cylinder, or 'h' for hemisphere. Following the shape character are numbers for the panel position, where these depend on the shape.

For 'r', enter the axis number that the rectangle is perpendicular to, preceded by a '+' if the panel front faces the positive axis and a '-' if it faces the negative axis (these signs must be entered); then enter the coordinates of a corner point; then enter the dimensions of the rectangle in sequential order of the axes, omitting the one that it is perpendicuar to. These dimensions are better called displacements because they are added to the corner that is entered, so they may be positive or negative. For example, for a square in a 3-D system that is perpendicular to the *y*-axis, has sides of length 10 and is centered about the origin, enter: "`panel r +1 -5 0 -5 10 10`". This same square could be entered as "`panel r +1 5 0 5 -10 -10`" , or with other descriptions. A rectangle is always perpendicular to an axis.

For 't', enter the coordinates of the corners of the triangle. This is one number for 1-D; 4 for 2-D, and 9 for 3-D. For 1-D, the front of the triangle always faces the positive axis; rectangles are completely equivalent and more versatile. For 2-D, triangles are really lines and the front side of the line is the side on the right when traveling in sequential order of the points that are entered. For 3-D, the triangle front is determined by the winding direction of the corners: if one is facing the front, the points wind counterclockwise. Unlike rectangles, triangles do not have to be perpendicular to axes.

For 's', enter the coordinates of the sphere center followed by the sphere radius and some drawing information. For 1-D, the center coordinate is a single number and the radius is entered next. For 2-D, the center coordinates are 2 numbers and then enter the radius followed by the number of sides on the polygon that should be drawn to represent the circle. For 3-D, the center coordinates are 3 numbers and then enter the radius, followed by the number of slices (longitude lines) and stacks (latitude lines) that are used for drawing the sphere. In the 2-D and 3-D cases, the drawing entries are used only for drawing; the circle or sphere functions as an accurate smooth shape. For all dimensions, enter a positive radius to have the front of the surface on the outside and a negative radius for it to be on the inside.

For '`c`', enter the coordinates of the cylinder-axis start point and the cylinder-axis end point, then the radius, and then drawing information if appropriate. Cylinders are not permitted in 1-D. In 2-D, two numbers give the start point and two give the end point, followed by the radius. No drawing information is needed. In 3-D, enter three numbers for the start point, three for the end point, the radius, and then the number of slices and the number of stacks.

For '`h`', enter the coordinates of the hemisphere center, the radius, and then the vector that points straight out of the hemisphere. Hemispheres are not permitted in 1-D. In 2-D, the center coordinates are 2 numbers, the radius is 1 number, the outward vector is 2 numbers, and finally enter the number of slices. For 3-D, the center is 3 numbers, the radius is 1 number, the outward vector is 3 numbers, and then enter 2 numbers for the numbers of slices and stacks. The outward pointing vector does not need to be normalized to unit length.

`end_surface`

End of a block of surface definitions. Surface statements are no longer recognized but other simulation statements are.

2.4 Runtime commands

Commands are stored in a queue, which is checked and executed just before the simulation starts, during the simulation, and just after it ends. All commands are declared in the configuration file using one of the forms shown above, where the final string is the actual command text. In some cases, the command text allows additional commands to be entered as well, allowing conditional expressions. Following is a list of possible command strings.

The command queue was changed slightly for version 1.55. In the new version, commands of type '`b`', '`a`', '`@`', and '`i`' are unchanged and run at the requested times. However commands of type '`e`' and '`n`' now run at exactly every iteration and every *n*'th iteration respectively, without round-off error.

Simulation control commands

`stop`

Stop the simulation.

`pause`

This puts the simulation in pause mode. If opengl graphics are used, continuation occurs when the user presses the spacebar. When graphics are not used, the user is told to press enter.

`keypress` *char*

Send a signal to the graphics manipulation component of the program to execute the behavior that would occur when a key is pressed. For the arrows, and shift-arrows, the character should be r for right, l for left, u for up, d for down, and the respective upper case characters for the shift-arrows.

<u>File manipulation commands</u>

`overwrite` *filename cmd*

Erase the output file called *filename* and then run command *cmd*.

`incrementfile` *filename cmd*

A new output file is created based upon the *filename*. The first time this is called the filename is appended with a "`_001`", which is then incremented with subsequent calls to "`_002`", and so on. These numbers precede any suffix on the filename.

<u>Conditional commands</u>

`ifno` *name cmd*

Run command *cmd* if no molecule of type *name* remains.

`ifless` *name num cmd*

Run command *cmd* if there are less than *num* molecules of type *name* remaining.

`ifmore` *name num cmd*

Run command *cmd* if there are more than *num* molecules of type *name*.

<u>System manipulation commands</u>

`pointsource` *name num* pos_0 pos_1 … pos_{dim}

Create *num* new molecules of type *name* and at location *pos*.

`volumesource` *name num* $pos_{0,\mathrm{low}}$ $pos_{0,\mathrm{high}}$ $pos_{1,\mathrm{low}}$ $pos_{1,\mathrm{high}}$ … $pos_{dim,high}$

Create *num* new molecules of type *name* and within the location bounded by pos_{low} and pos_{high}.

`killmol` *name*

Kill all molecules of type *name*.

`killmolinsphere` *name surface*

Kill all molecules of type *name* that are in any sphere that is a panel of surface *surface*. If *name* is "all" then every molecule type is killed. If *surface* is "all" then every surface is scanned.

`equilmol` $name_1$ $name_2$ *prob*

Equilibrate these molecules. All molecules of type $name_1$ and $name_2$ will be randomly replaced with one of the two types, where type $name_2$ has probability *prob*.

`replacexyzmol` *name* pos_0 pos_1 … pos_{dim-1}

If there is a non-diffusing molecule at exactly position *pos*, it is replaced with one of type *name*. This command stops after one molecule is found.

`replacevolmol` *name1 name2 frac* $pos_{0,\mathrm{low}}$ $pos_{0,\mathrm{high}}$ $pos_{1,\mathrm{low}}$ $pos_{1,\mathrm{high}}$ … $pos_{dim-1,\mathrm{high}}$

Fraction *frac* molecules of type *name1* in the volume bounded by pos_{low}, pos_{high} are replaced with ones of type *name2*.

`modulatemol` $name_1$ $name_2$ *freq shift*

Modulates molecules of types $name_1$ and $name_2$, just like `equilmol`, but with a variable probability. Every time this command executes, any of the two types of molecules in the system are replaced with a molecule of type $name_1$ with probability cos(*freq***t*+*shift*), where *t* is the simulation time, and otherwise with a molecule of type $name_2$.

`react1` *name rxn*

All molecules of type *name* are instantly reacted, resulting in the products and product placements given by the unimolecular reaction named *rxn*. Note that *name* does not have to be the normal reactant for reaction *rxn*.

`setrateint` *rxn rate*

Sets the internal reaction rate of the reaction named *rxn* to *rate*. See the description above for `rate_internal` for the meanings of *rate* for the different reaction orders.

`excludebox` *xlo xhi*
`excludebox` *xlo xhi ylo yhi*
`excludebox` *xlo xhi ylo yhi zlo zhi*

This keeps all molecules from entering a rectanguloid box within the system volume. Use the first form for one dimension, the second for two dimensions, and the third for three dimensions. Molecules that start within the box can stay there, but any molecule that tries to diffuse into the box is returned to its location at the previous time step. This command needs to be run at every time step to work properly.

`excludesphere` *x rad*

`excludesphere` *x y rad*
`excludesphere` *x y z rad*

This keeps all molecules from entering a sphere within the system volume. Use the first form for one dimension, the second for two dimensions, and the third for three dimensions; the coordinates given are the sphere center and *rad* is the sphere radius. Molecules that start within the sphere can stay there, but any molecule that tries to diffuse into the sphere is returned to its location at the previous time step. This command needs to be run at every time step to work properly.

`includeecoli`

An *E. coli* shape is defined as a cylinder with hemispherical endcaps, where the long axis of the bacterium extends the length of the *x*-axis within the system walls and the radius of both the cylinder and the endcaps is half the spacing between the walls that bound the *y*-axis. This command moves any molecule that diffuses out of the *E. coli* shape back to its location at the previous time step, or to the nearest surface of the *E. coli* if it was outside at the previous time step as well. This command does not need to be run at every time step to work properly. This only works with a 3 dimensional system.

System observation commands

`molcount` *filename*

Each time this command is executed, one line of display is printed to the listed file, giving the time and the number of molecules for each molecular species. The ordering used is the same as was given in the `names` command.

`listmols` *filename*

This prints out the identity and location of every molecule in the system to the listed file name, using a separate line of text for each molecule.

`listmols2` *filename*

This is very similar to `listmols` but has a slightly different output format. Each line of text is preceded by the “time counter”, which is an integer that starts at 1 and is incremented each time the routine is called. Also, the names of molecules are not printed, but instead the identity numbers are printed.

`listmols3` *name filename*

This is identical to `listmols2` except that it only prints information about molecules of type *name*.

`molpos` *name filename*

This prints out the time and then the positions of all molecules of type *name* on a single line of text, to the listed filename.

`molmoments` *name filename*

This prints out the positional moments of the molecule type given to the listed file name. All the moments are printed on a single line of text; they are the number of molecules, the mean position vector (*dim* values), and the variances on each axis and combination of axes (dim^2 values).

`savesim` *filename*

This writes the complete state of the current system to the listed file name, in a format that can be loaded in later as a configuration file. Note that minor file editing is often desirable before simulating a file saved in this manner. In particular, the saved file will declare its own name as an output file name, which will erase the configuration file.

`meansqrdisp` *name filename*

This function is used to measure mean square displacements (diffusion rates) of molecules of type *name*, printing the results to *filename*. When it is first invoked, it records the positions of all molecules of type *name*. Then, and every subsequent time it is called, it compares the current positions of all molecules that still exist to the old ones, calculates the average squared displacement, and prints the time and that number to a single line in the output file.

2.5 Sample configuration file

The following sample file executes a Lotka-Voltera reaction scheme, using parameters that are essentially the same as those used by Gillespie in his classic paper (*J. Phys. Chem.* 81:2340-2361, 1977). Using a vaguely ecological concept, R stands for rabbit and F stands for fox. The reactions are

$R \rightarrow 2\,R$

$R + F \rightarrow F$

$F \rightarrow$ nothing

The simulation is run in three dimensions with periodic boundary conditions.

```
# Simulation file for Lotka-Voltera reaction

graphics opengl
graphic_iter 5
accuracy 5
# rand_seed 5

dim 3
```

```
names R F
max_mol 20000
molperbox 1

difc R 100
difc F 100
color R 1 0 0
color F 0 1 0
display_size R 2
display_size F 3

time_start 0
time_stop 100
time_step 0.001

low_wall 0 -100 p
high_wall 0 100 p
low_wall 1 -100 p
high_wall 1 100 p
low_wall 2 -10 p
high_wall 2 10 p
mol 1000 R u u u
mol 1000 F u u u

cmd b pause
cmd e ifno R stop
cmd e ifno F stop

start_reaction
order 1
max_rxn 2
reactant R Rmultiply      # R -> 2R
rate Rmultiply 10
product Rmultiply R + R
reactant F Fdie            # F -> 0
rate Fdie 10
end_reaction

start_reaction
order 2
max_rxn 1
reactant R + F Feat     # R+F -> 2F
rate Feat 8000
product Feat F + F
end_reaction

end_file
```

3. Graphics

3.1 Graphics manipulations

The output graphics window displays the simulation volume with individual molecules. Essentially no viewing manipulations are possible with one or two dimensional simulations, although several are available with three dimensional ones. The display can be manipulated with keyboard input:

Key press	dimensions	function
space	1,2,3	toggle pause mode between on and off
Q	1,2,3	quit
T	1,2,3	save image as TIFF file
0	1,2,3	reset view to default
arrows	3	rotate object
shift + arrows	1,2,3	pan object
=	1,2,3	zoom in
-	1,2,3	zoom out
x,y,z	3	rotate counterclockwise about object axis
X,Y,Z	3	rotate clockwise about object axis

These are typically entered with actual key presses during the simulation run, although it is also possible to add them to the run-time command interpreter with the command `keypress`.

3.2 Saving images and movies

Assuming *Smoldyn* has been compiled with both the OpenGL library (needed for graphics) and with Sam Leffler's *libtiff* library, it is possible to save TIFF images. This can be done by pressing 'T' during a simulation to save a single snapshot. Alternatively, the configuration file statements `tiff_name`, `tiff_iter`, `tiff_min`, and `tiff_max` can be used to save a sequence of images that can then be compiled into a movie. Compiling is quite easy with Apple's QuickTime Pro, although there are probably other methods as well. Note that the image that is saved is a direct copy of the current graphics window, including window sizing, image rotation, and all other changes made by the user. Also, the saved TIFF won't be current if the graphics window wasn't updated.

4. Algorithm details

4.1 Binding and unbinding radii

For every bimolecular reaction, *Smoldyn* has to calculate the correct binding radius from the reaction rate that is given in the configuration file. Also, for every reaction that leads to multiple products, *Smoldyn* has to determine the correct unbinding radius, using whatever product parameter is supplied, if any. While these binding and unbinding radii are well defined microscopic parameters (at least within the context of the analytical model system that is simulated), the meanings of the experimental rate constants, as well as those given in the configuration file, are not nearly as well defined. Instead, those rate constants depend on the conditions under which they were measured. *Smoldyn* accounts for this by attempting to guess the experimental conditions, using a process described here. If *Smoldyn*'s guess is correct, the simulated reaction rates should exactly match the experimental rates (not including edge effects, which are typically negligible unless one reactant is fixed at or near an edge).

The product parameters are:

<u>Use these if reversible reactions were measured at equilibrium</u>

`p`	probability of geminate reaction (ϕ).
`x`	maximum probability of geminate reaction (ϕ_{max}).
`r`	unbinding radius relative to binding radius (σ_u/σ_b).
`b`	fixed length unbinding radius (σ_u).

<u>Use these if reversible reactions were measured with all product removed as it was formed</u>

`q`	probability of geminate reaction (ϕ).
`y`	maximum probability of geminate reaction (ϕ_{max}).
`s`	unbinding radius relative to binding radius (σ_u/σ_b).
`o`	fixed offset of products, rotationally randomized (σ_u).
`f`	fixed offset of products, not rotationally randomized (σ_u).
`i`	reaction is declared irreversible (σ_u=0).

In all cases, *Smoldyn* assumes that rate constants were measured using an effectively infinite amount of reactants that were started well mixed and that then were allowed to react until either an equilibrium was reached for reversible reactions, or a steady-state reaction rate was reached for irreversible reactions. Only in one of these cases is mass action kinetics correct and is the rate constant actually constant. The precise experimental assumptions are clarified with the following examples.

1. $A+B\rightarrow C$

The rate constant is assumed to have been measured at steady state, starting with a well-mixed system of A and B. No product parameter is required. At steady-state, the simulation matches mass action kinetics.

2. $X\rightarrow A+B$

There is no bimolecular reaction, so no binding radius is calculated. The default unbinding radius is 0, although it is possible to define a different one. If the product parameter is `p`, `q`, `r`, or `s`, an error is returned due to the lack of a binding radius. If it is not given or is `i`, `x`, or `y`, the unbinding radius is set to 0. If it is `b`, `f`, or `o`, the requested separation is used. At steady-state, the simulation matches mass action kinetics.

3. $A+B\leftrightarrow C$

If the reversible parameter is `p`, `x`, `b`, or `r`, the forward rate constant is assumed to have been measured using just this system of reactions after the system had reached equilibrium. The product parameter is used to yield the correct probability of geminate recombination if possible, or the desired unbinding radius. In this case, the simulation matches mass action kinetics at equilibrium. If the product parameter is `q`, `y`, `s`, `o`, `f`, or `i`, then it is assumed that the forward rate constant was measured at steady-state and with all C removed as it was formed, thus preventing any geminate reactions. The unbinding radius is set as requested, using the binding radius if needed. In this case, the simulated forward reaction rate is higher than requested due to geminate rebindings.

4. $A+B \leftrightarrow C \rightarrow Y$

The second reaction is ignored for determining parameters for A+B. Instead, the first reaction is considered as though the rates were determined experimentally using just the system given in example 3. If the product parameter is `p`, `x`, `r`, or `b`, the simulated reaction rate for the forward reaction $A+B \rightarrow C$ will be lower than the requested rate because there are fewer geminate reactions than there would be with the equilibrium system. Alternatively, it will be higher than the requested rate if the product parameter is `q`, `y`, `s`, `o`, `f`, or `i`, because there are some geminate reactions.

5. $X \rightarrow A+B \rightarrow C$

The binding radius for the second reaction is treated as in example 1, without consideration of the first reaction. The unbinding radius for the first reaction is found using the binding radius of the second reaction. Here, product parameters `p` and `q` are equivalent, `x` and `y` are equivalent, and `r` and `s` are equivalent. The actual reaction rate for the second reaction, found with a simulation, will be higher than the requested value due to geminate rebindings that occur after the dissociation of X molecules.

6. $X \rightarrow A+B \leftrightarrow C$

The $A+B \leftrightarrow C$ binding and unbinding radii are treated as in example 3. Another unbinding radius is required for the first reaction, which is found as in example 5, using the binding radius from the second reaction. Mass action kinetics are not followed.

7. $X \leftrightarrow A+B \leftrightarrow C$

The binding radii and unbinding radii for each bimolecular reaction are found as in example 3, independent of the other bimolecular reaction. The simulated rates may be different from those requested because of differing unbinding radii.

8. $X \rightarrow A+B \rightarrow C$, $A+B \rightarrow D$

The binding radii for the two bimolecular reactions are each found as in example 1. The unbinding radius for the first reaction cannot be determined uniquely, because the two forward reactions from A+B are equivalent and are likely to have different binding radii. *Smoldyn* picks the binding radius for the first forward reaction that is listed. Thus, if the product parameter for dissociation of X is `p`, the requested geminate rebinding probability will be found for the reaction A+B→C, but a different value will be found for the reaction A+B→D.

9. C↔A+B↔C

This reaction scheme might represent two different pathways by which A and B can bind to form an identical complex. However, *Smoldyn* cannot tell which reverse reaction corresponds to which forwards reaction. Instead, for both determining the binding and unbinding radii, it uses the first reverse reaction that is listed.

The general principle for calculating binding radii is that *Smoldyn* first looks to see if a reaction is directly reversible (*i.e.* as in example 3, without any consideration of reaction network loops or other possible causes of geminate reactions). If it is and if the reversible parameter is `p`, `x`, `r`, or `b`, then the binding radius is found under the assumption that the rate constant was measured using just this reaction, at equilibrium. If not, or if the reversible parameter is `q`, `y`, `s`, `o`, `f`, or `i`, then *Smoldyn* calculates the binding radius with the assumption that the rate constant was measured using just that reaction at steady-state and with all product removed as it is formed.

Unbinding radii typically require a reversible parameter (except as in example 2). If the parameter is `b`, `o`, or `f`, the requested unbinding radius is used. If it is `i`, the unbinding radius is set to 0. Otherwise, it can only be calculated with the knowledge of the binding radius. If the reaction is directly reversible, the binding radius for the reverse reaction is used. If it is not directly reversible but the products can react, as in examples 5, 6, and 8, then the binding radius for the first reaction that is listed is used.

4.2 Time steps

The simulated time in *Smoldyn* increases with discrete increments. However, a major focus of program design has been to make it so that results are indistinguishable from those that would be obtained if the simulated time increased continuously. This goal cannot be achieved perfectly. Instead, the algorithms are written so that the simulation approaches the Smoluchowski description of reaction-diffusion systems as the time step is reduced towards zero, and so it maintains as much accuracy as possible for longer time steps. This topic is discussed in detail in the research paper "Stochastic simulation of chemical reactions with spatial resolution and single molecule detail" by myself and Dennis Bray (*Physical Biology* 1:137-151, 2004). Some more discussion of this topic is given here.

In concept, the system is observed at a fixed time, then it evolves to some new state, then it is observed again, and so forth. A complication is that commands allow the user to manipulate the system at fixed times; it is typically best for these manipulations to immediately precede observations. For example, if a command states that some collection of molecules should be removed at time *t*, then an observation that is also at time *t* should show that they have been removed. This leads to the following sequence of program operations:

```
-------------- time = $t$ --------------
    manipulate system
    observe system
    diffuse molecules
    surface interactions
    reactions
        0th order reactions
        1st order reactions
        2nd order reactions
        add reaction products to system
    fix any product surface interactions
------------- time = $t + \Delta t$ -------------
```

To follow this scheme, manipulation commands should be entered before observation commands (the other order is possible as well if observations are desired before manipulations). After commands are run, graphics are displayed to OpenGL if that is enabled. The evolution over a finite time step starts by diffusing all mobile molecules. In the process, some end up across the walls of the boundary and others are within the binding radii of other reactants. Wall collisions are treated by reflecting molecules back into the simulation volume (for reflective boundaries). Next, reactions are treated in a semi-synchronous fashion. They are asynchronous in that all zeroth order reactions are simulated first, then unimolecular reactions, and finally bimolecular reactions. With bimolecular reactions, if a molecule is within the binding radii of two different other molecules, then it ends up reacting with only the first one that is checked, which is arbitrary. Reactions are synchronous in that reactants are removed from the system as soon as they react and products are not added into the system until all reactions have been completed. This prevents reactants from reacting twice during a time step and it prevents products from one reaction from reacting again during the same time step. As it is possible for reactions to produce molecules that are outside the system walls, those products are then reflected back into the system. At this point, the system has fully evolved by one time step. All molecules are inside the system walls and essentially no pairs of molecules are within their binding radii (the exception is that products of a bimolecular reaction with an unbinding radius might be initially placed within the binding radius of another reactant).

Each of the individual routines that is executed during a time step exactly produces the results of the Smoluchoski description, or yields kinetics that exactly match those that were requested by the user. However, the simulation is not exact for all length time steps because it cannot exactly account for interactions between the various phenomena. For example, if a system was simulated that only had unimolecular reactions and the products of those reactions did not react, then the simulation would yield exactly correct results using any length time step. However, if the products could react, then there are interactions between reactions and there would be errors. In this case, the error arises because *Smoldyn* does not allow a molecule to be in existance for less than the length of one time step.

4.3 Reactions near surfaces

Smoldyn does not treat reactions near walls or surfaces any differently from other reactions. Moreover, if walls are reflective and a reactant happens to be less than a binding radius from a wall, then part of the "binding volume" is inaccessible to other potential reactants; again, there is no special treatment for this.

When molecules have excluded volume, even inert impermeable surfaces can affect the local concentrations of chemicals. The obvious effect is that a molecule cannot be closer to a surface than its radius, leading to a concentration of zero closer than that. In a mixture of large and small molecules, Brownian motion tends to push the large molecules up against surfaces while the small molecules occupy the center of the accessible volume, creating more complex concentration effects. These effects do not occur when excluded volume is ignored, as it is in *Smoldyn*, in which case surfaces do not affect local concentrations.

While surfaces do not affect concentrations of non-reacting molecules, they do affect reaction rates. Consider the reaction A+B→C, where A is fixed and B diffuses. If essentially all A molecules are far from a surface, the diffusion limited reaction rate is found by solving the diffusion equation for the radial diffusion function (RDF) with the boundary conditions that the RDF approaches 1 for large distances and is 0 at the binding radius (see the paper by myself and Dennis Bray titled "Stochastic simulation of chemical reactions with spatial resolution and single molecule detail"). This leads to the Smoluchowski rate equation

$$k = 4\pi D\sigma_b$$

However, for an A molecule that is near a surface, an additional boundary condition is that the gradient of the 3 dimensional RDF in a direction perpendicular to the surface is zero at the surface. This makes the solution of the reaction rate sufficiently difficult that I have not attempted to solve it, but the result is different from the simple result given above. This surface effect is an issue whenever the A molecule is within several binding radii of a surface and is especially pronounced when it is closer to the surface than its binding radius. For cases in which the A molecule is more than one binding radius from the surface, B molecules are going to take longer than usual to reach the region between the A and the surface, leading to a decreased reaction rate. It is suspected that the reaction rate decreases monotonically as the A molecule approaches and then crosses a surface.

A special case that can be solved exactly occurs when the A molecule is exactly at the surface, such that half of the binding volume is accessible to B molecules and half is inaccessible. Now, the RDF inside the system volume is identical to the RDF for the case when the A molecule is far from a surface. The logic is to assume that this is true and to then observe that it already satisfies the additional boundary condition. Using this RDF, the diffusive flux is half of the diffusive flux for an A molecule far from a surface, because only half of the binding surface is exposed to the system. Thus, the diffusion limited reaction rate for the situation in which a reactant is fixed exactly at a surface is

$$k = 2\pi D\sigma_b$$

The situation changes some when simulation time steps are sufficiently long that rms step lengths are much longer than binding radii. Now, the probability of a reaction occuring during a time step is a function of only the binding volume. Thus, there are no surface effects at all when an A molecule is fixed anywhere in the simulation volume that is greater than or equal to one binding radius away from a surface. As the A molecule is moved closer to the surface, the reaction rate decreases in direct proportion to the binding volume that is made inaccessible to B molecules. An especially easy situation is that when the A molecule is exactly at the surface, the reaction rate is half of its value when the A molecule is far from a surface, which is the same as the diffusion limited result.

These results can be turned around to solve for the binding radius. If the reaction is diffusion limited, the binding radius should double when a reactant is placed exactly at the surface to maintain the same reaction rate. If it is activation limited, the binding radius should increase by $2^{1/3}$ to maintain the same reaction rate. As usual though, the binding radius is more closely related to the fundamental physical properties of the molecule than is the rate constant, so it is essential to consider the experimental conditions that were used for measuring the rate constant.

In conclusion, reaction rates are reduced near surfaces and the effect is different for diffusion limited and activation limited reactions. However, for both cases, and almost certainly for all cases in between, the reaction rate is exactly half when an A molecule is fixed at a surface, compared to when it is far from a surface. A few tests with *Smoldyn* using the files *reactW#*, described below, suggested that these surface effects are likely to be minimal for most situations, although it is good to be aware of their potential. The exception is that there are large surface effects when molecules are fixed with a significant portion of the binding volume outside the simulation volume.

4.4 Absorbing surfaces

As described in Andrews and Bray (*Phys. Biol.* 2004), a molecule is absorbed by an absorbing surface if it is decided that the molecule contacted the surface during the preceding time step. The molecule clearly contacted the surface if its postion after the time step is on the other side of the surface than it was before the time step. It is also possible for the molecule to stay on the same side, but to have contacted the surface during the time step. The probability of the latter case is

$$\exp\left[-\frac{l_1 l_2}{D\Delta t}\right]$$

where l_1 and l_2 are the perpendicular distances to the surface before and after the time step, respectively. This probability is implemented in Smoldyn to yield absorption probabilities that are accurate for any length time step and on a planar surface.

As of version 1.72, absorption to walls, which is now only possible if no other surfaces are defined, is simulated with the above equation. However, absorption to other surfaces do not account for these indirect collisions because they were difficult to code, increased computational load significantly, and made essentially no difference to results.

While it has not been implemented yet, a recent paper by Erban and Chapman (Phys. Biol., submitted 2006), presents the probability of the adsorption to a surface that is merely sticky and not totally absorbing. The stickiness coefficient is *K*, which is 0 for a reflective surface and infinite for a totally absorbing surface. If it is determined that surface contact occured during a time step, using the method just described, then the probability of adsorption should be

$$\mathrm{Prob}(\mathrm{adsorption} \mid \mathrm{contact}) = \frac{K\sqrt{\pi \Delta t}}{2\sqrt{D}}$$

This also applies to porous surfaces and will be implemented soon.

5. Tests of *Smoldyn*

5.1 Initial test: *bounce1*, *bounce2*, and *bounce3*

The simplest test is just to make sure that the *Smoldyn* applicaton is able to launch and run properly, using a very simple configuration file. These tests were also useful for getting the graphical output to work properly. Each file just shows a collection of molecules that bounce around inside the system walls. *bounce1* works in one dimension, *bounce2* in two dimensions, and *bounce3* in three dimensions. When running these, note that various keys can be used to control the graphics. These keys are: the arrows to rotate; shift + arrows to pan; '`=`' and '`-`' to zoom; '`x`', '`X`', '`y`', '`Y`', '`z`', and '`Z`' to rotate about the object axes; the space bar for pause; '`0`' to reset the view, '`T`' to save a tiff image, and 'Q' to quit. Not all of these are functional for 1 or 2 dimensions.

5.2 Diffusion rates: *diffi* and *diffa*

diffi was used to quantitatively test isotropic diffusion by letting several collections of molecules diffuse outwards from the center of space. The moments of the molecular distributions are saved as functions of time. The zeroth moment is the number of molecules, which obviously ought to stay at a constant value with no fluctuations. This was verified. The first moment is a vector quantity representing the mean position of the set of molecules. Because of symmetry, its value should stay near the initial position (the origin), although fluctuations are expected. These fluctuations are given with the equation

$$|\mathrm{mean} - \mathrm{starting\ point}| \approx \sqrt{\frac{2Dt}{n}}$$

D is the diffusion coefficient, *t* is the simulation time, and *n* is the number of molecules. This equation agrees well with simulation data. The second moment is a matrix quantity which gives the variance on each pair of axes of the distribution of positions. For example, the variance matrix element for axes *x* and *y* is

$$v_{xy} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

The overbars indicate mean values for the distribution. Equations are analogous for other pairs of axes. Because diffusion on different axes is independent, the off-diagonal variances (v_{xy}, v_{xz}, and v_{yz}) are expected to be about 0, but with some fluctuations, which was verified. However, the diagonal variances (v_{xx}, v_{yy}, and v_{zz}) are each expected to increase as approximately

$$v_{xx} \approx v_{yy} \approx v_{zz} \approx 2Dt$$

This behavior was verified for the simulation data, using a variety of simulation time steps and diffusion coefficients. However, the fluctuations of the variances were not analyzed.

diffa was used to investigate anisotropic diffusion. In this case, the diffusion equation is

$$\dot{u} = \nabla \cdot \mathbf{D}\nabla u$$

u can be interpreted as either the probability density for a single molecule or as the concentration of a macroscopic collection of molecules. **D** is the physical diffusion matrix, which is the square of the matrix that is entered in the configuration file (matrix square roots can be calculated with MatLab, Mathematica, or other methods). If **D** is equal to the identity matrix times a constant, *D*, the equation reduces to the standard isotropic diffusion equation. Using the file *diffa*, it was verified that the mean position for anisotropic diffusion is the initial position using diffusion matrices with and without off-diagonal elements. Using molecules that diffused in just the *x-z* plane, the variance on each axis was confirmed to be nearly equal to 2*Dt*, using the *D* value for each axis. It was also verified that the more complicated diffusion matrix yielded qualitatively reasonable diffusion, although the variances were not checked quantitatively.

5.3 Zeroth order reaction rates: *zeroreactf*, *zeroreactm*, *zeroreacts*

The zeroth order reaction nothing→A proceeds with the mass action rate equation

$$n(t) = n(0) + kt$$

$n(0)$ and $n(t)$ are the initial and time dependent numbers of A molecules and k is the zeroth order reaction rate. Using the file *zeroreactf*, *zeroreactm*, and *zeroreacts* (fast, medium, and slow) it was confirmed that the simulation results conform closely to the theoretical result, using rates ranging from 100 molecules per simulation time step to 0.01 molecules per time step. The simulation showed fluctuations in the production rates, as expected, but these were not analyzed.

5.4 Unimolecular reaction rates: *unireact1*, *unireactn*

unireact1 was used to check unimolecular reaction rates using a wide range of reaction rates, and thus a wide range of reaction probabilities in each time step. Each molecular species defined in the configuration file has a single unimolecular reaction pathway and simply goes away when it reacts. These chemical equations are each equivalent to simply A→nothing. The theoretical rate equation is

$$n(t) = n(0)e^{-kt}$$

$n(t)$ is the number of molecules remaining after time t, $n(0)$ is the initial number of molecules, and k is the first order rate constant. It was confirmed that simulation data agreed well with the equation using reaction probabilities that ranged from 0.001 to 0.632 per time step.

unireactn tests reaction probabilities using multiple reaction mechanisms. For convenience in analysis, the reactant concentration is kept constant by having the reactant as a reaction product. The reactions are

$A \rightarrow A+B$	$k_B = 2$
$A \rightarrow A+C$	$k_C = 1$
$A \rightarrow A+D$	$k_D = 0.5$

The system is started with only A molecules, so the theoretical number of B molecules as a function of time is

$$n_B(t) = k_B n_A t$$

Analogous equations hold for C and D. Simulation results closely matched these theoretical equations when a small time step was used. However, a large time step leads to a relatively large probability that an individual A molecule will react by one of the three pathways during the time step. If this probability is p (the sum of the probabilities for each of the three reaction pathways), then the probability that an A should react twice during the same time step is proportional to p^2. In the simulation, a molecule can only react once during a time step, leading to errors whenever p^2 is large, and hence when p is large.

5.5 Bimolecular reactions, different reactants: ***react ABs***, ***react ABm***, ***react ABf***

The reaction A+B→C is easy to analyze using mass action kinetics with the condition that there are the same numbers of A and B molecules initially. The solution for the number of A molecules (or B molecules) as a function of time is

$$n(t) = \left(\frac{1}{n(0)} + \frac{kt}{V} \right)^{-1}$$

reactABs, *reactABm*, and *reactABf* (slow, medium, and fast) were used to test the simulated reaction rate. In all cases, but especially with the fast reaction rate, the simulated rate is faster initially than the analytical rate because the simulation starts with molecules randomly distributed whereas the analytical result assumes a steady-state distribution. However, after enough time passes for a steady state distribution to be formed, the simulated results agree quite well with the analytical results. These files examine reaction rates that have ratios of mutual rms step lengths to binding radii ranging from 0.146 (*bireactABf*, diffusion limited) to 2.15 (*bireactABs*, activation limited).

5.6 Bimolecular reactions, same reactant: ***reactAAs***, ***reactAAm***, and ***reactAAf***

The reaction A+A→C was investigated as well. Using mass action kinetics, the analytical solution for the number of A molecules as a function of time is

$$n(t) = \left(\frac{1}{n(0)} + \frac{2kt}{V} \right)^{-1}$$

This was tested using the files *reactAAs*, *reactAAm*, and *reactAAf*. All files agreed well with the analytical equation. The configuration file ratios of mutual rms step length to binding radius ranged from 0.145 for the fast version (diffusion limited) to 1.67 for the slow version (activation limited). As before, the simulated reaction rate was faster initially than the analytical rate because of different initial molecule distributions. However, the rates agreed well after the simulation distribution had time to reach a steady-state profile.

5.7 Reactions near walls: ***reactW1***, ***reactW2***, ***reactW3***, ***reactW4***

In principle, an impermeable surface should not change the local concentration of a reactant, although it can affect reaction rates. This was investigated using the same system as for the A+B→C reaction shown above, except that all A molecules were fixed near the surfaces of the simulation volume. The binding radius for all files is 0.763 and the rms step length of the B molecules is 0.632, leading to reactions that are intermediate between diffusion and activation limited. In the file *reactW1*, the A molecules are positioned 5 units inside reflective walls, which is large compared to the rms step length or binding radius. Kinetics are quite different from those described in section 4.5 because of the correlated A molecule positions and surface effects, although an analytical equation was not derived. Changing the distance from the walls to 1 unit in *reactW2*, which is now close to the rms step length and binding radius, makes essentially no difference in the output. In *reactW4*, the A molecules are placed on the walls and the boundaries are made periodic, which again has no effect. Other tests involved increasing or decreasing the time steps by a factor of 10 for these files, which also had no significant effects. Thus, surface effects are minimal for reaction rates when the entire binding volume is accessible to diffusing molecules.

In *reactW3*, the A molecules are placed on the walls and the boundaries are reflective so that only half of the binding volume is accessible. The reaction rate in this case should be exactly half of the rate for the other files, although it was consistently found that the ratio of the rates is closer to 0.55, for a wide range of time step lengths. The cause of this difference has not been identified.

5.8 Reversible reactions: *equil*

Reversible reactions involve geminate recombination issues, as discussed in section 3. The accuracy of reversible reaction rates was investigated with the configuration file *equil*, in which an equilibrium is set up for the reaction A+B↔C. From standard chemistry, the equilibrium constant is related to the ratio of product to reactant concentrations and to the ratio of the forward to reverse rate constants,

$$K = \frac{n_C V}{n_A n_B} = \frac{k_f}{k_r}$$

V is the total system volume. The configuation file *equil* starts with equal numbers of A and B molecules and no C molecules. Using the above equation and this starting point, the solution for the equilibrium number of A molecules is

$$n_{\mathrm{A}} = \frac{-V + \sqrt{V^2 + 4Kn_{\mathrm{A}}(0)V}}{2K}$$

$n_{\mathrm{A}}(0)$ is the initial number of A molecules. It was verified that the simulation result approached this value. As usual, fluctuations were not analyzed.

5.9 Simple reaction networks: *lotvolt*

The file *lotvolt* just runs a simple Lotka-Volterra system of reactions to make sure that the various components of *Smoldyn* work together. This configuration file is identical to the one shown in section 2.5. Results were not analyzed quantitatively.

5.10 Surfaces: *surf1*, *surf2*, *surf3*

Surfaces in 1, 2, and 3 dimensions are demonstrated and tested with these three files. They include all surface panel shapes and all surface types. These types are reflecting, transparent, and absorbing. For the first time, version 1.72 is nearly perfect about not letting molecules leak through surfaces that are supposed to be reflective.

6. Copyright and Citation

I expect to maintain a working copy of the program indefinitely. The current download site for *Smoldyn* is http://genomics.lbl.gov/~sandrews/index.html, where the program may be obtained for free. If improvements are made to the code or bugs are fixed, then I would appreciate a copy of the modified source code. If you find any bugs in the code, please let me know! My e-mail address is ssandrews@lbl.gov.

If *Smoldyn* is used to a significant extent, it may be appropriate to cite or acknowledge its use.

7. Acknowledgements

I started writing *Smoldyn* while I was a post-doc in Dennis Bray's laboratory at the University of Cambridge and funded by NIGMS grant GM64713. Additions were made during my next post-docin Adam Arkin's laboratory at the Lawrence Berkeley National Laboratory, where I was funded by the Genomes to Life Project of the US Department of Energy and an NSF post-doctoral fellowship in biological informatics. Suggestions and comments from Karen Lipkow, Dennis Bray, and Adam Arkin are appreciated. Prior to my starting *Smoldyn*, I attended an *M-Cell* workshop taught by Joel Stiles and Tom Bartoll, where I learned many of the concepts that became incorporated in the program.